Digital Image Processing Laboratory: Image Halftoning

March 29, 2021

1 Introduction

An 8-bit monochrome image allows 256 distinct gray levels. Modern computer monitors generally support the display of such images, however some other rendering technologies allow for much fewer gray levels. At the far end of the spectrum are devices, such as printers, that can only display two levels, black or white, for a monochrome image.

This lab will introduce a useful area of image processing called *halftoning*, which is the conversion of a grayscale image into a binary image. The key in this application is to exploit properties of the human visual system to give the impression of a continuous tone image even though only two levels are present in the rendering. Halftoning is required in several electronic applications such as facsimile (FAX), electronic scanning and copying, and laser and inkjet printing.

This lab will emphasize two halftoning techniques known as *ordered dithering* and *error diffusion*. Exercises covering these techniques will all be performed in Python. **An important note about rendering:** The halftone images you produce in this lab need to represented exactly pixel-by-pixel to get the proper visual effect. Therefore, when viewing images within Python, use *interpolation='none'* in *plt.imshow* in order to map 1-to-1 the image pixels to display pixels. Otherwise, the image will likely interpolate on your display and obscure the intended binary result. For the same reason, your halftone results should be written out to TIFF files(using PIL as below) and submitted independently in a zip file along with your report.

```
from PIL import Image
img_out = Image.fromarray(x)
img_out.save('img_out.tif')
```

To help facilitate comparison between the various halftone images in this lab, we will use an image fidelity metric that incorporates a simple model of the human visual system. This requires some background, so we will begin with a short discussion of image fidelity.

Questions or comments concerning this laboratory should be directed to Prof. Charles A. Bouman, School of Electrical and Computer Engineering, Purdue University, West Lafayette IN 47907; (765) 494-0340; bouman@ecn.purdue.edu

2 Image Fidelity Metrics

As we explore halftoning methods in this lab, we will also assess quantitatively how well the halftone images reproduce the original grayscale images. We will see that simply computing the average squared pixel error is a fairly useless metric in this application because it fails to take into account the spatial blurring response of the human visual system. This blurring property is precisely what halftoning methods exploit to make a binary pattern appear as a continuous gray tone.

The numerical comparison we will use between a halftone image, $g(x, y)$, and the original grayscale image, $f(x, y)$, is represented in Figure 1. A very important property of the human

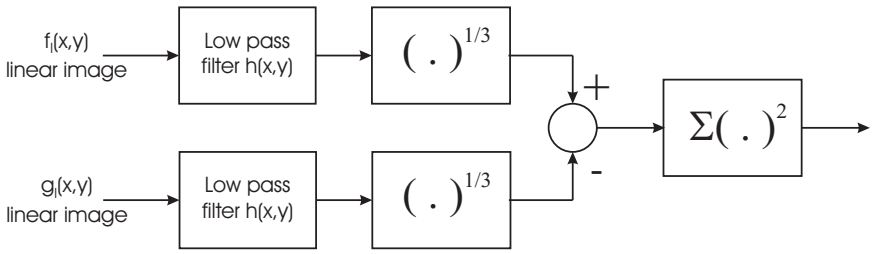

Figure 1: Block diagram for a simple image fidelity metric

visual system is the spatial blurring response, represented here by the low pass filters. This is an optical blurring of *light energy*, therefore the scale of the input images needs to be *linear in energy*. This means that if the images we want to compare are encoded in a *gamma-corrected* scale, denoted $f_g(x, y)$ and $g_g(x, y)$, then we first need to undo the gamma correction with the transformation,

$$f_l(x, y) = 255 \left(\frac{f_g(x, y)}{255} \right)^{\gamma} \tag{1}$$

and similarly for $g_g(x, y)$. Equation (1) assumes the original image pixel values are in the range [0,255]. The topic of *gamma correction* was discussed in depth in a previous laboratory so we will limit further discussion of that here.

The purpose of the pixel-wise $(\cdot)^{1/3}$ operations prior to taking the difference is to weight pixel differences in terms of *contrast* rather than direct light energy. Simply put, the visual system is more sensitive to differences in light energy in darker regions than in lighter regions, and the cube-root operation places a scaling that will weight these differences in brightness equally in a perceptual sense.

3 Thresholding and Random Noise Binarization

The simplest method of converting a grayscale image to a binary image is by *thresholding*, i.e. a two-level (one-bit) quantization. Let $f(i,j)$ be a grayscale image, and $b(i,j)$ be the corresponding binary image based on simple thresholding. For a given threshold T, the binary image is computed as

$$b(i,j) = \begin{cases} 255 & \text{if } f(i,j) > T \\ 0 & \text{else} \end{cases} . \tag{2}$$

(a) (b)

Figure 2: (a) Original grayscale image. (b) Binary image produced by simple fixed thresholding.

Figure 2 illustrates the conversion to a binary image by thresholding, using $T = 127$. It can be seen that the binary image is not "shaded" properly–an artifact known as *false contouring*. This often occurs when quantizing at low bit rates (one bit in this case), because the quantization error is highly dependent upon the input signal. If one reduces this dependence, the visual fidelity of the binary image can be greatly enhanced.

One way to reduce the signal/error dependence is to add uniformly distributed white noise to the input image prior to quantization. Specifically, to each input pixel of the grayscale image, $f(i,j)$, add an independent *uniform[-A,A]* random number, and quantize the result according to equation (2). An illustration of this approach is shown in Figure 3, where the additive noise is uniform over $[-128, 128]$. Notice that even though the resulting binary image is somewhat noisy, the false contouring has been dramatically reduced, and with enough blurring (e.g. looking at it from a distance) it gives the impression of having several gray levels.

3.1 Exercise: Thresholding

1. Download the image *house.tif* from the lab web page. Apply the simple thresholding of equation (2) to the image, using $T = 127$.

Figure 3: Random noise binarization.

2. Compute the *root mean square error* (RMSE) between the original and binary images, defined by

$$\text{RMSE} = \sqrt{\frac{1}{NM}\sum_{i,j}\{f(i,j) - b(i,j)\}^2} \tag{3}$$

where NM is the total number of pixels in each image. **Note:** In this calculation, be sure that both images f and b are of type *double*. If either image is of type *uint8*, differences outside the range [0,255] will clip.

3. Compute the *image fidelity*, which we will define by

$$\text{fidelity} = \sqrt{\frac{1}{NM}\sum_{i,j}\{\tilde{f}(i,j) - \tilde{b}(i,j)\}^2} \tag{4}$$

where $\tilde{f}(i,j)$ and $\tilde{b}(i,j)$ are the original and binary images transformed by the systems illustrated in Figure 1. The steps for this transformation include the following:

   (a) Un-gamma correct $f(i,j)$ and $b(i,j)$ using equation (1) and $\gamma = 2.2$. Note the un-gamma transformation does not actually change the binary image. (Why?)

   (b) Low-pass filter $f(i,j)$ and $b(i,j)$ using a 7×7 Gaussian filter, defined by the following point spread function,

$$h(i,j) = \begin{cases} C\ \exp(-\frac{i^2+j^2}{2\sigma^2}) & \text{for } |i| \le 3 \text{ and } |j| \le 3 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

   where $\sigma^2 = 2$, and C is a normalizing constant such that $\sum_{i,j} h(i,j) = 1$.

   (c) Apply the transformation $y = 255(x/255)^{1/3}$ to each pixel value of the filtered f and b images.

You will have to compute the fidelity for several results going forward, so you should write a function `fid=fidelity(f,b)` to avoid replicating the code.

Section 3.1 Report:

- Hand in the original image and the result of thresholding.
- Submit the computed RMSE and fidelity values.
- Hand in the code for your *fidelity* function.

4 Ordered Dithering

The goal in halftoning is to give the impression of grayscale tones while using only black and white pixels. Although the random thresholding technique described in Section 3 can produce this effect, it is not often used in real applications since it yields very noisy results. In this section, we will describe a better class of halftoning techniques known as *ordered dithering*.

Because the human visual system tends to average a region around a pixel instead of sensing each pixel individually, we can create the illusion of many gray levels in a binary image that in actuality only contains two gray levels. Using 2×2 binary pixel grids, we can represent 5 different "effective" intensity levels, as illustrated in Figure 4. Similarly for 3×3 grids, we can represent 10 distinct gray levels. In *dithering*, we replace blocks of the original image with these types of binary grid patterns.

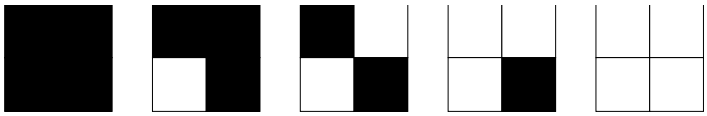

Figure 4: Five different patterns of 2×2 binary pixel grids.

Remember from Section 3 that false contouring artifacts can be reduced if we reduce the dependence between the quantization error and the original signal. We illustrated that adding uniform noise to the monochrome image can achieve this decorrelation. Equivalently, one could use a variable threshold in the quantization process, which is related to the approach used in dithering.

Ordered dithering consists of comparing blocks of the original image to a 2-D grid of thresholds called a *dither pattern*. Each element of the original block is quantized according to the corresponding threshold value in the dither pattern. The values in the dither matrix are fixed, but are typically different from each other. Because the threshold changes between adjacent pixels, some decorrelation from the quantization error is achieved.

A dither matrix can also be defined by a so called *index matrix*. The index matrix determines the order in which dots are "turned on" (change from white to black) as the image becomes darker (greater absorptance). For the example in Figure 4, the corresponding index matrix is given by

$$I_2(i,j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix} \tag{6}$$

where 0 indicates the first pixel to turn on, and 3 indicates the last pixel to turn on. This index matrix is a special case of a family of dither matrices first defined by Bayer [1], which are defined recursively by,

$$I_{2n} = \begin{bmatrix} 4 * I_n + 1 & 4 * I_n + 2 \\ 4 * I_n + 3 & 4 * I_n \end{bmatrix} \tag{7}$$

where I_{2n} is the new $(2N) \times (2N)$ matrix and I_n is the old $N \times N$ matrix. This operation can be performed with the following Python command.

```
I2N=np.block([[4*IN + 1,4*IN + 2],[4*IN + 3,4*IN]])
```

For each index matrix, there is a corresponding threshold matrix which is used to halftone the image. The threshold matrix can be determined from the index matrix $I(i,j)$ by the relationship,

$$T(i,j) = 255\frac{I(i,j)+0.5}{N^2} \quad , \quad 0 \leq i,j \leq N-1 \tag{8}$$

where N^2 is the total number of elements in the matrix. This produces thresholds evenly spaced between 0 and 255. The halftoning is performed by thresholding each pixel in the original image according to the values in $T(i,j)$. Since the image is usually much larger than the threshold matrix, the dither pattern is repeated periodically, or *tiled*, across the full image. Specifically we can write the operation as,

$$b(i,j) = \begin{cases} 255 & \text{if } f(i,j) > T(i \bmod N, j \bmod N) \\ 0 & \text{else} \end{cases} \quad . \tag{9}$$

Figure 5 shows the result of Bayer dithering using a 4×4 pattern. It is clear that the halftone image provides decent detail rendition. However, the down side is that square grid patterns are typically visible in the halftone image.

Figure 5: The halftone image produced by Bayer dithering of size 4.

4.1 About Halftoning Gamma Corrected Images

As described above, the goal of halftoning is to produce a pattern of black and white pixels that will be perceived as a continuous gray level due to the blurring effect of the human visual system. This blurring is the averaging of *light energy* over a set of black and white pixels, and the perceived light energy will be proportional to the fraction of white pixels in the local region. Therefore the original grayscale image that the halftoning is attempting to reproduce must also be scaled proportional to light energy for the perceived gray levels to match. This is a very important point because most digital images are encoded in a gamma-corrected scale, and such images need to be transformed to linear scale (using (1) for example) *prior to encoding the halftone image.*

If you remove the gamma correction from a grayscale image, it will likely display too dark on a computer monitor, but that is due to the internal mapping of the computer monitor–it is *not* a reflection of the true light energy represented in the image file. The intended light energy will be displayed properly when the image is encoded in a gamma corrected scale, and when the gamma value matches the gamma of the monitor. Note this gamma response of the monitor *has no effect* on a halftone image because the image contains only the minimum and maximum display values.

4.2 Exercise: Ordered Dithering

1. Download the image *house.tif* from the lab web page, and read it into your Python code.
2. This image has been gamma corrected, so produce a linear-scale version by applying the transformation in equation (1), with $\gamma = 2.2$. Also retain the original version for display purposes.
3. Create Bayer threshold matrices of sizes 2×2, 4×4, and 8×8.
4. Generate three different halftone images for *house.tif* by applying these three dither patterns *to the linear-scale version*. When displaying the halftone images in Python, remember to use *interpolation='none'* in *plt.imshow* to prevent interpolation of your binary image. Note that it is sometimes better to view the halftone results from a slight distance.
5. Export the halftone results to TIFF files using *imwrite*.
6. For each of the three halftone images, compute the RMSE and fidelity between the halftone and the original image. Use the same procedure described in Section 3.1.

Section 4.2 Report:
Submit the following with your report:

1. The three Bayer index matrices of sizes 2×2, 4×4, and 8×8.
2. The three halftoned images produced by the three dither patterns.
3. The RMSE and fidelity for each of the three halftoned images.

5 Error Diffusion

Another class of halftoning techniques are called *error diffusion*. In this method, the pixels are quantized in a specific order (raster ordering[1] is commonly used), and the residual quantization error for the current pixel is propagated (diffused) forward to local unquantized pixels. This keeps the local average intensity of the binary image close to the original grayscale image.

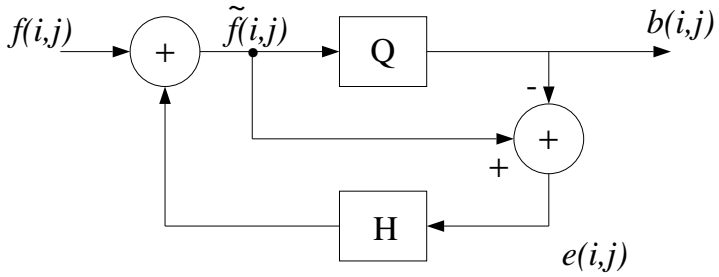

Figure 6: Block diagram of the error diffusion method.

Figure 6 is a block diagram that illustrates the error diffusion algorithm. The current input pixel $f(i,j)$ is modified by adding certain past quantization errors, producing a modified input $\tilde{f}(i,j)$. This pixel is then converted to a binary value by the quantizer Q, using some threshold T. The current error $e(i,j)$ is defined as

$$e(i,j) = \tilde{f}(i,j) - b(i,j) \tag{10}$$

where $b(i,j)$ is the quantized binary pixel value.

The error $e(i,j)$ of quantizing the current pixel is diffused to "future" pixels by means of a two-dimensional weighting filter $h(i,j)$, known as the *diffusion filter*. The process of modifying an input pixel by past errors can be represented by the following recursive relationship.

$$\tilde{f}(i,j) = f(i,j) + \sum_{k,l \in S} h(k,l)e(i-k, j-l) \tag{11}$$

A very popular error diffusion method, proposed by Floyd and Steinberg [2], uses the diffusion filter shown in Figure 7. Since the filter coefficients sum to one, the local average value of the quantized image will be equal to the local average grayscale value.

Figure 8 shows a halftone image produced by Floyd and Steinberg error diffusion. Compared to ordered dither halftoning, the error diffusion method can be seen to have better contrast performance. However, error diffusion tends to create "streaking" artifacts, also known as *worm* patterns.

[1]Raster ordering of an image orients the pixels from left to right, and then top to bottom. This is similar to the order that a CRT scans the electron beam across the screen.

	•	7/16
3/16	5/16	1/16

Figure 7: Point Spread Function of the error diffusion filter proposed by Floyd and Steinberg.

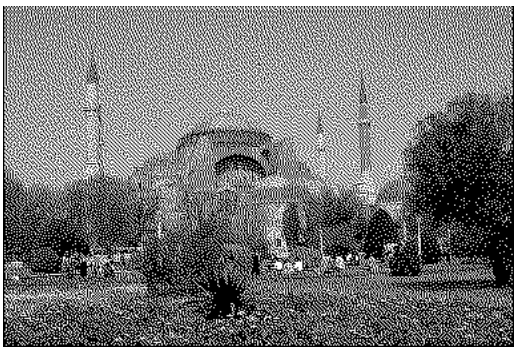

Figure 8: A halftone image produced by Floyd and Steinberg error diffusion method.

5.1 Exercise: Error Diffusion

Apply the error diffusion technique to *house.tif* using a threshold $T = 127$ and the diffusion filter in Figure 7. As in the previous exercise, the algorithm should be applied to a *linear-scale* version of *house.tif*.

A straight forward implementation of the error diffusion algorithm is detailed in the following steps, performed on each pixel in raster order:

1. Initialize an output image matrix with zeros.
2. Quantize the current pixel to 0 or 255 using using the threshold T, and place the result in the output matrix.
3. Compute the quantization error by subtracting the binary pixel from the grayscale pixel.
4. Add scaled versions of this error to "future" pixels of the original image, according to the diffusion filter of Figure 7.
5. Proceed to the next pixel.

Display the result in Python using *plt.imshow* (remember to use *interpolation='none'*), and compare this to the original image. Again, it is best to view the results from a slight distance. Write out the halftone result to a TIFF using *PIL.Image.save*.

Compute the RMSE and fidelity between the error diffusion result and the original image, using the same procedure described in Section 3.1.

Section 5.1 Report:
Submit the following:

1. Your error diffusion Python code
2. The error diffusion result
3. The RMSE and fidelity of the error diffusion result
4. Finally, tabulate the RMSE and fidelity for the simple thresholding, ordered dithering, and error diffusion results. Comment on your observations of both the RMSE and fidelity for the different methods. Relate these metrics to the observed visual quality.

References

[1] B. E. Bayer, "An optimum method for two-level rendition of continuous-tone pictures," in *IEEE International Conference on Communications*, vol. 1, June 11-13 1973, pp. 11–15.

[2] R. W. Floyd and L. Steinberg, "An adaptive algorithm for spatial greyscale," *Journal of the Society for Information Display*, vol. 17, no. 2, pp. 75–77, 1976.